Языки для математиков

Математику можно считать синонимом слова программирование, потому что, создавая любую программу, сначала ее нужно переводить на язык математики, потом только на определённый язык программирования. Не зная математику и физику, серьёзные программы просто невозможно сделать.

— из интернет-форума

Достойно удивлению, почему столь многие из нас, изучив отвратительные структуры типов традиционных языков программирования с помощью изящного инструментария, разработанного Д.Скоттом, пассивно сохраняют верность этим структурам вместо того, чтобы энергично искать новые структуры.

— Дж.Бэкус

Размышления о том, почему важна математика для программирования …
Логическое мышление, абстракции ….

В математике два фундаментальных понятия: множество и функция.

Во всех языках программирования есть функции. Например, функция, вычисляющая значения квадратичного трёхчлена Эйлера, записывается так: f(x)=x*x+x+41. На Паскале надо написать:

```
function f(x:integer): integer;
begin
      f  := x*x + x + 41
end;
```

Здесь возникает вопрос, а зачем такие сложности (относительно, конечно)? Не было бы достаточно в программе обойтись одной строкой «f(x)=x*x+x+41» для определения функции? И, действительно, такие языки есть. Даже в одном из первых языков ФОРТРАН такая строка определяла *оператор-функцию*, которая размещалась в начале программы. Параметр «x» являлся вещественной переменной, так как действовало правило умолчания для определения типа переменной по первой букве имени. Если написать «n» вместо «x», то была бы функция от целой переменной.

Второй вопрос связан именно с типами. В языках со строгой типизацией (в том числе и в Паскале) для приведенного примера приходится писать другую функцию с другим именем, если мы хотим делать вычисления, например, с вещественными значениями.

И, наконец, самый важный вопрос. Любая ли функция (в том же Паскале) является функцией в математическом плане? Рассмотрим следующий фрагмент

```
y1:= f1(1); y2:= f1(1);
writeln(y1,y2);
```

Получим ли мы одинаковые значения? Ответ: не обязательно. Например, для функции, заданной следующим образом, значение y2 окажется на 1 больше, чем y1:

```
function f1(x:integer): integer;
begin
      a := a+1;  {глобальная переменная}
      f1  := x + a
end;
```

Использование в теле функции глобальных переменных дает так называемый *побочный эффект*. Такой эффект во многих случаях нежелателен, хотя бы потому, что не позволяет просто так перенести код функции из одной программы в другую.

В некоторых языках (Ада, Оберон) функции называют процедурами, возвращающими значения с заменой ключевого слова «function» на «procedure», В Аде есть функции (function), но компилятор проверяет отсутствие побочных эффектов.

Создатель Фортрана, Дж.Бэкус, в своей тьюринговской лекции 1977 года говорил Существует и целый класс языков, называемых функциональными, в которых функции играют существенную роль. Для одного из них, языка Haskell, проведем небольшой экскурс.

На странице www.haskell.org/hugs/pages/downloading.htm скачайте файл WinHugs-Sep2006.exe и установите систему. Создайте файл func.hs со строкой «f(x)=x*x+x+41». Запустите winhugs.exe и откройте созданный файл. Система напишет, что она загрузила и оттранслировала файл, и выдаст приглашение для ввода команд или вызова функций:

```
Hugs> :load "D:\\...\\func.hs"
Main>
```

Теперь вы можете вводить арифметические выражения, и система будет работать как калькулятор. Вызов описанной в файле функции: «f(0)» или «f 0», т.е. скобки не нужны! Не нужны они и в определении функции.

В функции не указываются типы аргументов и результата. Она может применяться к любым типам, для которых определены в данном случае операции «*» и «+». Даже если ввести, например, тип для представления матриц, и определить для них умножение и сложение, то функция f будет применима и для них. В системе есть модуль Ratio для работы с дробями. Разместите в первой строке файла оператор import Ratio и перегрузите функцию (команда «reload» или пункт меню). Теперь функция применима для дробных значений:

```
Main> f 1%3
43 :% 3
```

Чтобы получить значений сразу для нескольких значений, например для всех чисел от 0 до 10, вызывается функция map:

```
Main> map f [0..10]
[41,43,47,53,61,71,83,97,113,131,151]
```

Первый аргумент – это имя нашей функции, второй задает диапазон изменения чисел. Обратите внимание, что в списке результатов все числа простые.

Вас приятно удивит, что целые числа неограниченной размерности. Можно посчитать 2^1000 и даже 2^1000000 (в последнем числе 301030 цифр!).

Понятие функции – основное в Haskell. И кроме него практически ничего нет. **Нет операций**. Запись x*x есть другая форма от вызова функции (*), т.е. функцию f можно определить так:

```
f x = (+) ((+) ((*) x x) x) 41
```

А любую функцию от двух переменных можно использовать как операцию. Например, пусть есть функция nod – наибольший общий делитель. Тогда ее вызов можно записать так:

```
k = n `nod` m
```

Нет переменных! Единственные имена, с которым мы имеем дело в теле функций, это формальные параметры. И нет как таковой последовательности вычислений. Если встречается строка «x=1», то это не переменная, и не константа. Это функция без параметров, имеющая постоянное значение.

Нет циклов! Но как же записать функцию вычисления факториала или того же и НОД? Используя *рекурсию* (функция вызывает сама себя):

```
fact n = if n==1 then 1 else n*(fact (n-1))
```

или еще проще без if:

```
fact1 1 = 1
fact1 n = n*(fact (n-1))
```

Функция вычисления НОД:

```
nod n m | n==m = n
nod n m | n>m   = nod (n-m) m
nod n m | n<m   = nod n (m-n)
```

В последних примерах функция задается несколькими «уравнениями». При первом вызове и последующих рекурсивных вызовах для исполнения выбирается подходящее уравнение, начиная с первого.

В языке две структуры данных: список и кортеж. Список заключается в квадратные скобки и состоит из значений одного типа, кортеж – в круглых скобках и значения могут быть разных типов. Список задается простым перечислением, указанием границ или с помощью конструктора. Пример конструктора, задающего множество чисел, делящихся на 3:

```
[k | k <- [1..10], mod k 3 == 0]
```

Для математика эта запись должна быть знакома: так определяются подмножества элементов с определенными свойствами:

$S_1 = \{k \mid k \in S, \mod_3 k = 0\}$

Определим функцию, возвращающую список делителей целого числа:

```
delims n = [k | k <- [1..n], mod n k == 0]
```

А теперь можно определить функцию, проверяющую является ли число простым (число, делящееся только на 1 и само число):

```
ifPrime n = delims n == [1,n]
```

Вернёмся к примеру функции, с которой мы начали. Чем интересен квадратичный трёхчлен Эйлера? Возьмем первые 40 значений функции для аргумента от 0 до 39 и проверим сколько среди этих значений простых чисел:

```
length (filter isPrime (map f [0..39]))
```

Получим, что все они простые! А если взять больший отрезок?

Простые числа давно интересуют математиков. Они пытались найти формулу, например, в виде многочлена с целыми коэффициентами от одной переменной, который давал бы все простые числа при подстановке целых чисел. Квадратичный трёхчлен Эйлера – одна из таких попыток. Оказалось, что таких многочленов не существует. Интересен вопрос, а может ли многочлен давать бесконечно много простых чисел. Ответа пока нет.

Но было доказано, что существует многочлен с несколькими переменными, положительные значения которого дают все простые числа (есть пример такого многочлена пятой степени с 26-ю переменными – *где-то было в Кванте, пока не нашел*)

Простые числа дают пищу и для программистов. Одна из таких задач – задача нахождения простых чисел Мерсенна (числа вида 2^n-1). С ними связаны и совершенные числа. Число совершенное, если оно равно сумме своих делителей. В древности знали только 2 числа: 6 и 28. Потом Евклид нашел 2 следующих: 496 и 8128. В XVI-веке

Катальди нашел еще 2 числа, В 1972 Эйлер нашел 8-е число Мерсенна 2147243647. И более через 100 лет русский священник нашел 19-значное число без всяких вычислительных приборов! Прорыв начался в 1952-м году с появлением компьютеров. В 1996 году был создан проект http://www.mersenne.org/ для нахождения чисел, в котором участвуют сотни тысяч человек. Каждый желающий может подключиться к сети проекта и начать проверку простоты чисел. Организация Electronic Frontier Foundation учредила премии за нахождения чисел Мерсенна с 1 млн. цифр, с 10 млн. цифр и т.д. Первую премию 50000$ получил индиец Н.Хаджратвала в 2000 году. Следующая премия была получена в 2008 году за число, содержащее 11185272 цифры, коллективом из Калифорнийского университета. Последнее 48-е число из 17425170 цифр было найдено в 2013 году. Кто найдет число с более чем 100 млн. знаков, получит 150000$.

В языке Haskell всего 24 ключевых слова, из которых половина известна по языкам Си и Паскаль:

as, case, class, default, do, else, if, in, import, then, of, type,
_, data, deriving, infix, infixl, infixr, instance let, module, newtype, qualified, where

1. Лекции лауреатов премии Тьюринга: Пер. с англ./Под ред. Р. Эшенхёрста. – М. : Мир, 1994.. – 560 с. : ил.
2.

Необработанное дополнение …
http://ega-math.narod.ru/Quant/Primes.htm#note9txt

http://ega-math.narod.ru/Quant/Primes.htm

Ряд известных авторов[11][12] выделяют в особую категорию «языки, наследованные от математики» (англ. mathematically-derived languages). Алан Кэй также отделяет языки, являющиеся «стилем во плоти» (crystalization of style) от прочих языков, являющихся «склеиванием возможностей» (agglutination of features)[13].

Это языки, семантика которых является непосредственным воплощением некой математической модели, незначительно адаптированной (без нарушения целостности) для того, чтобы быть более практичным языком для разработки реальных программ. Лишь некоторые языки попадают под эту категорию, большинство языков проектируются приоритетно исходя из возможности эффективной трансляции в машину Тьюринга, и имеют лишь некое подмножество в своём составе, воплощающее ту или иную математическую модель — от арифметики до средств параллелизма (например, Occam-?[en] — это Occam, дополненный набором конструкций, воплощающих \pi-исчисление)..

Примеры математически обоснованных языков и воплощаемых ими математических моделей:

Язык	Основы	Авторы, время создания, характеристика
Agda	Интуиционистская теория типов Мартин-Лёфа	
APL и его потомки (J, K)	оригинальная семантика, не имеющая названия, воплощающая нотацию Айверсона для исчисления массивов (часто встречается термин «*array languages*»)	Ken Iverson, Harvard University, 1957-1960 APL was originally intended not as a programming language but as a notation for the concise expression of mathematical algorithms. It went unnamed and unimplemented for many years. Finally a subset APL\360 was implemented in 1964. APL is an interactive array-oriented language with many

		innovative features, written using a non-standard character set. It is dynamically typed with dynamic scope. All operations are either dyadic infix or monadic prefix, and all expressions are evaluated from right to left. The only control structure is branch. APL introduced several functional forms but is not purely functional.
Coq	исчисление индуктивных конструкций	
Erlang	исчисление процессов (первоначально в форме модели акторов, позже также построено обоснование на π-исчислении)	
Forth	стековая машина и конкатенативный язык программирования	Charles H. Moore, 1960's An interactive extensible language using postfix syntax and a data stack. A program is a set of functions ("words") which are compiled by an outer interpreter into bytecodes. FORTH is small and efficient, but programs can be difficult to read. It was used first to guide the telescope at NRAO, Kitt Peak.
Haskell	теория категорий (включая «декартово замкнутую категорию», воплощающую лямбда-исчисление; категорию монад для моделирования побочных эффектов; расширение системы типов Хиндли-Милнера; систему родóв и др.)	
Joy	композиция функций и гомоморфизм (иначе говоря, чистый конкатенативный язык программирования, и, как следствие, чистый функциональный)	Manfred von Thun, La Trobe University, 1994 A purely functional language with postfix syntax.
Lisp	лямбда-исчисление Чёрча (в том числе язык S-выражений, воплощающий нотацию пар Чёрча)	John McCarthy et al, MIT, late 50's A symbolic functional recursive language based on lambda-calculus, used especially for AI and symbolic mathematics. Many dialects. Atoms and lists. Dynamic scope. Both programs and data are represented as list structure
Scheme	«облагороженный» диалект Лиспа (сильнее типизированный, в большей степенигомознаковый[I], ограничивающийся гигиеническими макроопределениями и соблюдающий числовую башню), дополненный нотацией продолжений)	Guy L. Steele & G. J. Sussman, 1975 A LISP dialect, small and uniform, with clean semantics. Scheme is applicative-order and lexically scoped, and treats both functions and continuations as first-class objects.
ML	типизированное лямбда-исчисление, то есть лямбда-исчисление, дополненное системой типов Хиндли-Милнера	Meta Language R. Milner et al, 1973 A strict higher-order functional language with statically-checked polymorphic types, garbage collection and a formal semantics. It started out as the metalanguage for the Edinburgh LCF proof assistant. (LCF="Logic for Computable Functions")
Prolog	исчисление предикатов	PROgrammation en LOGique Alain Colmerauer and Phillipe Roussel, University of Aix-Marseille, 1971. First implemented 1972 in ALGOL-W.

		Prolog was designed originally for natural-language processing. It uses LUSH (or SLD) resolution theorem proving based on the unification algorithm. There are no user-defined functions, and no control structure other than the built-in depth-first search with backtracking. Early collaboration between Marseille and R. Kowalski at the University of Edinburgh continued until about 1975. Programming in Prolog" William F. Clocksin & C. S. Mellish, Springer-Verlag, 1985.
Mercury	исчисление предикатов, дополненное системой типов Хиндли-Милнера	
Smalltalk	теория множеств (с соблюдением числовой башни)	Software Concepts Group, Xerox PARC, led by Alan Kay, early 70's Smalltalk took the concepts of class and message from SIMULA 67 and made them all-pervasive, the quintessential object-oriented language. Innovations included the graphical user interface: bitmap display, windowing system and use of mouse.
SQL	исчисление кортежей (вариант реляционного исчисления, в свою очередь основанного на исчислении предикатов первого порядка	
SGML и его потомки (HTML, XML)	нотация деревьев (важный случай графов)	
Unlambda	комбинаторная логика	David Madore "Your functional programming language nightmares come true." A bizarre functional language. An implementation of the lambda-calculus without the lambda operation, relying entirely on the K and S combinators. No variables, data structures, loops or conditionals.
Многие языки	Регулярные выражения	
Рефал	оригинальная семантика Турчина, носящая название «*Рефал-машины*» или «*Рефал-автомата*», созданная на основе нормального алгоритма Маркова, воплощающая композицию теории автоматов, сопоставления с образцом и переписывания термов	Recursive Functions Algorithmic Language V. Turchin, Moscow, about 1972

Математическая логика
Комбинаторика
Теория графов
Теория алгоритмов
Теория автоматов
Теория множеств
Теория формальных грамматик
Булева алгебра
Логическое программирование

Функциональное программирование
?-исчисление
Теория искусственного интеллекта

https://ru.wikipedia.org/wiki/Языки_программирования